VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

REVIZNÍ SYSTÉM PRO LaTeX

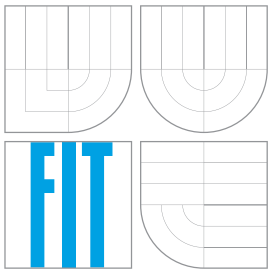
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAROSLAV DIVILA

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

REVIZNÍ SYSTÉM PRO LaTeX

REVISION SYSTEM FOR LaTeX

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAROSLAV DIVILA
AUTHOR

VEDOUCÍ PRÁCE — Ing. JIŘÍ ZUZAŇÁK
SUPERVISOR

BRNO 2010

Abstrakt

Tato práce se zabývá návrhem a implementací revizního systému pro sázecí systém LaTeX. Zaměřuje se zejména na návrh efektivní a úsporné metody pro uchovávání změn ve zdrojových souborech a operací nad nimi. To vše s podporou správy více uživatelů.

Abstract

This thesis deals with conception and implementation of revision system for typesetting system LaTeX. The work is focused on conception of effective and space-saving method for changes of source codes and operations with them. Everything in multiuser mode.

Klíčová slova

LaTeX, Revizní systém, Systém pro správu verzí

Keywords

LaTeX, Revision System, Version control system

Citace

Jaroslav Divila: Revizní systém pro LaTeX, bakalářská práce, Brno, FIT VUT v Brně, 2010

Revizní systém pro LaTeX

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jiřího Zuzaňáka.

. . . . . .
Jaroslav Divila
17. května 2010

Poděkování

Rád bych poděkoval panu Ing. Jiřímu Zuzaňákovi za vedení mé práce a odbornou pomoc.

Obsah

Kapitola 1

Úvod

S rozvojem počítačové technologie a vytvářením obsáhlých softwarových projektů, vznikala i potřeba ukládat si předchozí verze zdrojových souborů a v případě potřeby se k nim vrátit. Z toho důvodu začaly vznikat revizní systémy, které umožňovaly rychlý a efektivní přístup k předchozím verzím projektů, zjišťovaly rozdíly mezi jednotlivými změnami a dalšími způsoby usnadňovaly práci vývojářům. Dnes v době týmové spolupráce více vývojářů na jednom projektu, si práci bez podobného systému nelze ani představit.

Cílem této práce je navrhnout a implementovat revizní systém specializovaný na uchovávání změn ve zdrojových souborech LaTeXu s podporou správy více uživatelů, možností vracet se k předchozím verzím a odstraňovat jednotlivé změny.

Druhá kapitola této práce mapuje historii a vývoj revizních systémů od prvního až po současnost. A seznámí nás s vybranými zástupci různých typů revizních systémů. Třetí kapitola pojednává o technologiích, se kterými je nutno se seznámit před samotným návrhem aplikace. Ve čtvrté kapitole je popsán návrh metody uchovávání změn a revizí zdrojových souborů LaTeXu a návrh aplikace. Pátá kapitola popisuje implementaci aplikace a seznamuje nás s jejími funkcemi. Šestá kapitola zhodnocuje dosažené výsledky a nastiňuje další možný vývoj aplikace a její možná rozšíření.

Kapitola 2

Systémy pro správu verzí

Tato kapitola popisuje vybrané revizní systémy, od prvních průkopnických systému, až po dnešní moderní distribuované systémy. Při psaní této kapitoly jsem čerpal z [1, 2].

2.1 Lokální revizní systémy

Lokání revizní systémy jsou systémy, které nepodporují přístup po síti. Revize souborů se uchovávají pouze na lokálním disku počítače, na kterém je systém používán. Mezi tyto systémy patří dva nejstarší zástupci revizních systémů SCCS a RCS. V dnešní době jsou zastaralé a prakticky se nepoužívají.

2.1.1 SCCS

Prvním systémem pro správu verzí byl SCCS (Source Code Control System). Původně byl vyvinut v roce 1972 v Bell Labs Markem V. Rochkindem pro IBM System/370. Později byl přepsán pro UNIX a používán na počítači PDP-11, následně byl využíván v různých UNIXových distribucích a získal si velkou oblibu. Dnes je považován za zastaralý, ale jeho souborový formát se používá v několika dnešních revizních systémech, například v systémech BitKeeper či TeamWare.

Více o systému SCCS lze nalézt v [10].

2.1.2 RCS

RCS (Revision Control System) vytvořil v roce 1982 Walter F. Tichy na Pordue University, jako alternativu pro tehdy ještě populární SCCS. V současnosti je RCS součástí GNU projektu, avšak jeho podpora byla zastavena v roce 1995.

RCS pracuje pouze s jednotlivými soubory, proto se nehodí pro revizi celých adresářů či projektů. RCS, kromě registrace změny a návratu k stavu před změnou, umožňuje vypisovat zprávy o stavu a historii souboru, generuje popisy změn mezi libolnými verzemi souboru a přestože spolupráce více uživatelů se spíše nepoužívala, má k ní RCS prostředky, kterými jsou zámky. Při spolupráci více uživatelů si může jeden uživatel soubor ručně zamčít a ostatní s ním nemůžou pracovat, to znamená, že nemůže dojít ke kolizi.

Další informace o RCS jsou přístupné v [1, 9].

2.2 Centralizované revizní systémy

Centralizované revizní systémy mají obvykle architekturu klient – server. Úložiště revizí tzv. repozitář je vytvořen na serveru a uživatelé si stahují jeho pracovní kopie na lokální disky, kde je upravují a změny pak nahrávají zpět do repozitáře. Tento způsob je výhodnější pro přístup více uživatelů, pokud ovšem dojde k havárii repozitáře, může nastat problém. Identifikace revizí u těchto systémů je numerická. Dva neznámější zástupci jsou CVS a Subversion.

2.2.1 CVS

CVS (Control Versions System někdy také Control Versioning System) byl vyvinut ze systému RCS Dickem Grunem a jeho studenty v roce 1986. Do dnešní podoby začal CVS vyvíjet Brian Berliner s pozdějším přispěním Jeffa Polka a dalších spolupracovníků. V roce 1990 bylo CVS verze 1.0 vydáno pod společností Free Software Foundation pro vývoj a distribuci.

Na rozdíl od RCS pracuje CVS už s celým projekty a nabízí spolupráci více uživatelů po síti. Typicky má proto systém CVS architekturu klient – server, to kromě spolupráce více uživatelů, zajišťuje i to, že budou změny provedené kterýmkoli uživatelem konzistentní a budou distribuovány ostatním uživatelům.

Aby bylo možno pracovat s celými projekty, přichází CVS s konceptem tzv. repozitářů (repository). Repozitář je skupina souborů jednoho projektu a repozitáře jsou ještě děleny na moduly, které představují na sobě nezávislé části projektu. Každý repozitář má vlastní řízený přístup a je uložen ve formě souborů na souborovém systému serveru. Změny jsou uchovávány na úrovni verzí jednoho souboru ve stromové struktuře projektu. V kterémkoli okamžiku je možné vytvořit novou vývojou větev a samostatně ji rozvíjet nebo později opět sloučit. Uživatel si stahuje ze serveru pracovní kopii repozitáře na lokální disk, kde ji upravuje a následně odesílá změny zpět do repozitáře na serveru.

CVS má i nevýhody, mezi hlavní patří například to, že operace, při níž se nahrávají změny v souborech zpět do repozitáře (`commit`), je neatomická, to znamená že, každá změna se provádí zvlášť, a v případě, že operace skončí chybou, či je jinak přerušena, je v repozitáři uložena jen část změn, což vede k nekonzistentnímu stavu. Další nevýhoda, je malá podpora pro, zejména opakované, slučování větví. Při sloučení dvou větví se nezachovává historie obou větví, jen nová revize v jedné. Dále nelze kopírovat či přesouvat adresáře a práce s CVS je časově a prostorově náročná. Problémem je také mazání adresářů, přejmenovávání adresářů nebo souborů a neefektivní síťová komunikace.

Podrobnosti lze nalézt v [8] a [1] obsahuje vyčerpávající návod k použití a popis funkcionality CVS.

2.2.2 Subversion

Projekt Subversion nebo-li SVN začal v roce 2000 ve firmě CollabNet Inc. Cílem bylo vytvořit open-source revizní systém, který je podobný reviznímu systému CVS, ale opravuje jeho chyby a podporuje vlastnosti, které v CVS chybí.

Filosofií, technickými principy a stylem použití se Subversion od CVS příliš neliší. Oproti CVS existuje v systému Subversion více přístupových cest k repozitářům. SVN ukládá vždy jen informace o provedených změnách, takže má mnohem nižší prostorovou náročnost. Operace `commit` je atomická a pokud je přerušena neprojeví se v repozitáři žádná změna. Naopak slučování větví je stále problematické, stejně jako neefektivní síťová komunikace.

Více o systému Subversion je uvedeno v [7].

2.3 Distribuované revizní systémy

Distribuované jinak též decentralizované systémy se skládají z hlavního repozitáře a lokálních repozitářů, do kterých se hlavní repozitář klonuje. Z lokálních repozitářů si pak jednotlivý uživatelé stahují pracovní kopie jako u centralizovaných systémů. Lokální repozitáře zaručují samostatný paralelní lokální vývoj a hlavní repozitář se pak používá jen jako synchronizační bod vývojářů a pro jednoduchou výměnu změn v podobě oprav a podobě.

V následujícím textu si popíšeme tři zástupce distribuovaných systému, GNU Arch, systém SVK vystavěný na Subversion a BitKeeper.

2.3.1 GNU Arch

GNU Arch je první distribuovaný revizní systém, jeho základy položil v roce 2001 Thomas Lord. Příkaz pro manipulaci s GNU repozitáři je `tla`, což je akronym pro Tom Lord's Arch. V roce 2003 se stal součástí GNU Projektu.

GNU Arch je velmi flexibilní, zejména co se týče komunikace po síti, ke komunikaci lze použít prakticky jakýkoliv protokol pro přenos souborů. Bezproblému zvládá slučování větví, i slučování pouze vybraných revizí (cherrypicking) a to zásluhou pokročilého slučovacího algoritmus (merge star), který zajišťuje bezproblémové slučování i větví, které už byli dřívě sloučeny, nebo větví, které si někdy dříve vyměnily jen některé revize pomocí cherrypickingu.

Na druhou stranu je GNU Arch velmi pomalý, používá dlouhá a nepraktická jména souborů a revizí a nefunguje dobře v operačním systému Windows.

Další informace jsou dostupné v [11].

2.3.2 SVK

Systém SVK je založen na souborovém systému Subversion. Je napsán v jazyce Perl a hlavním autorem je Kao Chia-liang.

Se systémem Subversion má mnoho společných rysů, zvládá přesouvání a kopírování adresářů či verzování metadat. Ale podporuje i nové operace jako je práce offline, dále podporuje pokročilé slučovací algoritmy (merge star, cherrypicking) a je velmi rychlý, třikrát rychlejší než Subversion.

Více o systému SVK je uvedeno v [12].

2.3.3 BitKeeper

BitKeeper je systém, který je od základu vystavěn pro distribovaný vývoj. Je produktem firmy BitMover Inc. a jedná se o komerční software s uzavřeným zdrojovým kódem, který se sice může používat zdarma, ale nese to s sebou určitá omezení.

BitKeeper podporuje sady změny, ale zároveň si každý soubor uchovává i svou vlastní revizi. Největší důraz je kladen na větvení, slučování a zejména distribuovaný vývoj. Pracuje velice efektivně s více větvemi a používá vylepšený algoritmus pro řešení případných problému. I komunikace po síti je velice efektivní.

Kapitola 3

Použité technologie

Tato kapitola stručně popisuje technologie, které budou využity při implementaci aplikace, či technologie s nimiž sama aplikace pracuje.

3.1 LaTeX

Zadáním práce je vytvořit revizní systém pro LaTeX, při vytváření takového systému bude třeba navrhnout metodu udržování záznamu o změnách v LaTeXových dokumentech. Aby mohla být taková metoda správně a efektivně navržena je nejprve nutné se seznámit se strukturou zdrojových souborů LaTeXu a s LaTeXem obecně. O tom pojednává tato kapitola.

Při zpracovávání této kapitoly jsem čerpal z [5, 6].

3.1.1 Historie

Kolem roku 1983 vytvořil pro své potřeby profesor Donald E. Knuth sázecí systém TeX, který umožňoval jednoduchou sazbu textu a zejména matematických rovnic a zachovával vysokou typografickou kvalitu výsledného dokumentu. Nad tímto systémem vznikla celá řada nadstaveb usnadňující jeho použití běžným uživatelům. Nejznámější nadstavbou je systém LaTeX.

LaTeX vytvořil Leslie Lamport kolem roku 1994. Jde o balík maker, který umožňuje sázet dokumenty s vysokou typografickou kvalitou za pomoci předdefinovaných vzhledů.

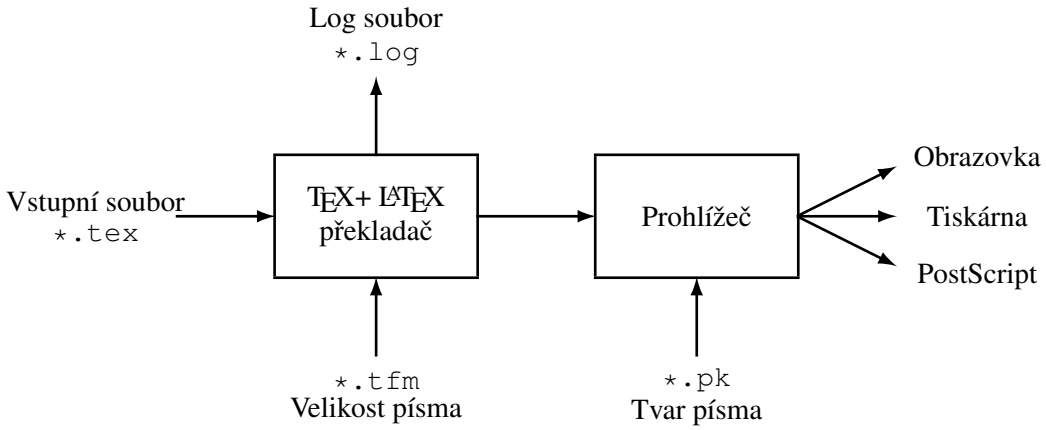
V současnosti pracuje tým LaTeX3, pod vedením Franka Mittelbacha, na sjednocení všech rozšiřujících verzí LaTeXu, které vznikly od vzniku LaTeXu 2.09. Tato nová verze nese název LaTeX 2_ε.

3.1.2 Princip funkce

LaTeX není běžný textový editor, ve kterém uživatel vidí vzhled dokumentu, tak jak bude vypadat, hned při psaní (tzv. WYSIWYG z anglického What you see is what you get – co vidíš je to co dostaneš.). V LaTeXu je obsah dokumentu spolu s příkazy pro formátování psán do vstupního souboru a až po zpracování tohoto souboru LaTeXem je možno výslednou podobu dokumentu zobrazit na obrazovku, či vytisknout (obrázek 3.1).

3.1.3 Syntaxe vstupního souboru

Vstupní soubor je textový ASCII soubor s koncovkou `*.tex`. Obsahuje text dokumentu, doplněný o příkazy LaTeXu, které určují jeho výslednou podobu.



Obrázek 3.1: Překlad vstupního souboru

V příkazech LaTeX rozlišuje velká a malá písmena. Příkazy se uvozují zpětným lomítkem a následuje jméno příkazu složené pouze z písmen, nebo se skládají ze zpětného lomítka a speciálního znaku. Příkazy končí mezerou nebo znakem, který není písmeno. Mezery a ostatní bílé znaky za příkazy LaTeX ignoruje. Některé příkazy vyžadují k své činnosti parametr, který se uvádí za příkaz do složených závorek. Některé příkazy umožnují zadat volitelný parametr, který se vkládá do hranatých závorek.

Všechny ostatní bílé znaky interpretuje LaTeX jako mezeru. Více bílých znaků za sebou interpretuje jako jednu mezeru a bílé znaky na začátcích řádků ignoruje.

Níže uvedené znaky jsou rezervované a mají v LaTeXu zvláštní význam. Vysázeny budou pouze v případě, že jsou uvedeny ve dvojici se zpětným lomítkem, např. `\$`.

```
$ & % # _ { } ~ ^
```

Komentáře jsou uvozeny znakem `%` a končí s koncem řádku.

3.1.4 Struktura vstupního souboru

Vstupní soubory LaTeXu mají pevně danou strukturu. Každý vstupní soubor musí začínat příkazem

```
\documentclass[option]{class}
```

Tento příkaz definuje třídu dokumentu a ve volitelném parametru lze nastavit formát stránky, velikost písma a pod. Následující část vstupního souboru se nazývá preambule. Preambule může obsahovat příkazy, které ovlivní vzhled celého dokumentu, uživatel zde může definovat vlastní příkazy a rozšířit funkce LaTeXu připojením balíků maker příkazem:

```
\usepackage[options]{package}
```

Dále následuje samotná textová část, která je uvozena příkazem

```
\begin{document}
```

a obsahuje text dokumentu s dalšími příkazy. Textová část je ukončena příkazem

```
\end{document}
```

tímto příkazem také končí celý vstupní soubor a cokoliv následujícího je LaTeXem ignorováno.

3.2 XML

XML (Extensible Markup Language) je obecný značkovací jazyk vytvořený konsorciem W3C. XML je zjednodušenou verzí staršího jazyka SGML a je navržený tak, aby informace v něm bylo možno spracovávat jako objekty, nikoli jako posloupnosti znaků. Jazyk XML umožnuje vytvářet vlastní značky, nezajímá se jak jsou data zobrazena, ale poskytuje informace o datech samotných, jejich struktuře a uspořádání. Data v dokumentu XML jsou čitelná jak pro lidi tak pro stroje. Především je jazyk XML určen pro výměnu dat mezi aplikacemi a publikování dokumentů, u kterých popisuje strukturu z hlediska věcného obsahu jejich částí.

Tuto kapitolu jsem zpracoval podle [3].

3.2.1 Vlastnosti

Jazyk XML je založen na jednoduchém otevřeném formátu, který není spjat s žádnou platformou. Dokumenty XML obsahují jednoduchý text a jsou zpracovatelné libovolným textovým editorem. XML používá implicitně znakovou sadu Unicode a tak je možné vytvářet XML dokumenty v různých světových jazycích. Současně je možné použít i jiné kódování než Unicode, musí však být určeno jaké.

Dokumenty vytvořené v XML mají vysoký informační obsah, do značek můžeme vkládat metadata v podobě atributů, které vyznačují význam různých částí textu.

3.2.2 Syntaxe

Aby byl dokument XML správně uspořádaný je nutné dodržet syntaktická pravidla stanovená konsorciem W3C. Níže je uveden příklad jednoduchého XML dokumentu.

```
<?xml version="1.0" enconding="UTF-8" standalone="yes"?>

<uzivatel id="01">
  <jmeno>Jaroslav</jmeno>
  <prijmeni>Divila</prijmeni>
  <login>xdivil00</login>
</uzivatel>
```

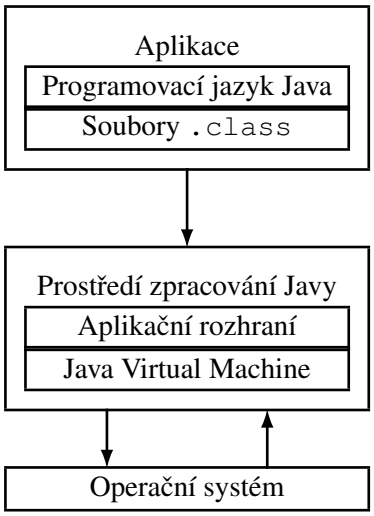
Na prvním řádku se nachází instrukce pro zpracování dokumentu, ta říká analyzátoru jaké verze XML dokument je, jaká je použita kódovací sada a zda dokument obsahuje odkazy na další soubory. Instrukce pro zpracování dokumentu začínají značkou `<?` a končí značkou `?>`. Každý dokument XML musí obsahovat kořenový element, který obsahuje všechny ostatní elementy. V našem případě je to element `<uzivatel>`. Elementy musí začínat počáteční značkou`<jmeno>` a končit koncovou značkou `</jmeno>`, vyjma prázdného elementu, který může být zapsán značkou `<name/>`. Element může obsahovat textová data nebo další elementy, elementy se ale nesmí překrývat. Elementy mohou obsahovat atributy, hodnoty atributů musí být uzavřeny v jednoduchých či dvojitých uvozovkách.

3.3 Java

Jako implementační jazyk aplikace jsem zvolil jazyk Java. Java je objektově orientovaný programovací jazyk. Jeho hlavní výhodou je platformová nezávislost, ta je způsobena speciálním virtuální prostředím (JVM - Java Virtual Machine), které odděluje kód aplikace od operačního systému, na kterém je spoušten. Architektura se skládá ze čtyř částí:

- Programovací jazyk Java
- formát souboru `.class`
- aplikační programové rozhraní Javy
- virtuální stroj Javy

Zdrojový kód jazyka Java je překládán do bytového kódu, zapisovaného do souborového formátu `.class`. Tyto soubory jsou pak spouštěny v prostředí virtuálního stroje Javy. Prostředí pro zpracování jazyka Javy je znázorněno na obrázku 3.2.



Obrázek 3.2: Prostředí pro zpracování jazyka Java

Dalším důvodem pro zvolení jazyka Java, byla možnost využití objektového modelu DOM pro práci s XML dokumenty a využití open-source knihovny SVNkit, pro komunikaci s SVN serverem.

3.3.1 Zpracování XML dokumentů v jazyce Java

V jazyce Java je možné využít pro zpracování XML dokumentů dva standardy, rozhraní SAX (Simple API from XML) a model DOM (Document Object Model).

Aplikační rozhraní SAX bylo vytvořeno jako výsledek fóra elektronické diskuze. Je určeno primárně ke čtení XML dokumentů, které zpracovává sekvenčně, není tedy nutné načíst celý XML dokument do paměti. Hlavní výhoda tohoto přístupu spočívá v tom, že i při zpracovávání velmi obsáhlých dokumentů spotřebovává minimum systémových prostředků.

Objektový model DOM byl navržen konsorciem W3C a je výkonější než SAX. Při zpracování XML dokumentu je celý dokument načten do paměti, kde je reprezentován jako hierarchicky uspořádaná kolekce objektů. Úpravou této kolekce lze pak upravovat obsah i strukturu dokumentu. Model DOM se proto používá k vytváření nových XML dokumentů a aktualizaci již vytvořený, je velmi všestraný a přizpůsobivý. Jeho nevýhodou je vyšší paměťová náročnost.

Podrobné informace pro zpracování XML dokumentů v jazyce Java je uvedeno v [3].

3.3.2 SVNkit

SVNkit je open-source knihovna, pro přístup k verzovacímu systému Subversion, napsaná čistě v jazyce Java. SVNkit implementuje všechny vlastnosti Subversion a poskytuje aplikační rozhraní pro přístup k pracovním kopiím a pro přístup a manipulaci s repozitáři.

High-Level API umožnuje přes třídu `SVNClientManager` přístup k velkému množství rozhraní, které implementují bežné operace se Subversion jako je stáhnutí aktuální verze repozitáře do pracovní kopie (`checkout`), aktualizace pracovní kopie (`update`), uložení změn do repozitáře (`commit`), dále umožnuje získání historie souborů, získání rozdílu mezi jednotlivými verzemi souborů a podobně.

Low-Level API pomocí třídy `SVNRepository` poskytuje přístup k práci s repozitáři přímo na úrovni protokolů SVN. Toho je možné využít, pokud máme speciální požadavky na funkcionalitu nebo efektivitu.

SVNkit poskytuje přístup k repositorům přes `http(s)://`, `svn(+ssh)://` a `file://` protokoly. Vzhledem k tomu, že je napsán čistě v Javě, je stejně jako Java multiplatformní, bylo popsáno, že pracuje na operačních systémech Windows, OSX, Linux, BSD a OpenVMS. SVNkit je kompatibilní s JDK 1.4.

Více informací je k nalezení na domovských stránkách projektu [4].

Kapitola 4

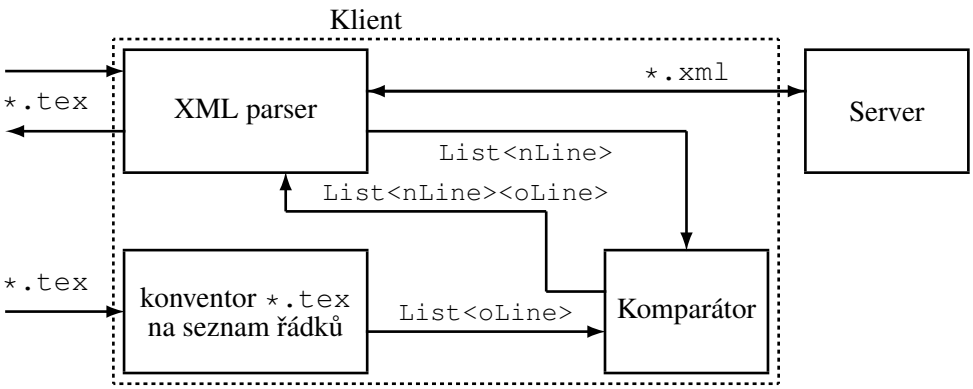
Návrh

V této kapitole se seznámíme s návrhem samotné aplikace a její funkcionality a s návrhem metody udržování záznamu o změnách ve zdrojových souborech LaTeXu.

4.1 Popis návrhu aplikace

Zadáním práce bylo vytvořit aplikaci pracující na bázi klient – server, která umožní archivaci zdrojových souborů LaTeXu, bude udržovat záznamy o změnách ve zdrojových souborech a umožní návrat k předchozím verzím. Aplikace také podporuje správu více uživatelů.

4.1.1 Klientská část



Obrázek 4.1: Blokové schéma aplikace

Aplikace bude mít tři základní funkce, ukládání nových projektů, aktualizace a uchovávání změn již uložených projektů a načítání uložených projektů a jejich verzí. Na obrázku 4.1 vidíme zjednodušené schéma celé aplikace.

Při ukládání nového projektu se v XML parseru vytvoří nový XML dokument stejného jména jako zdrojový soubor LaTeXu `*.tex` (dále jen zdrojový soubor) a po řádcích se do něj načte. Takto vytvořený XML dokument se odešle na server.

Aktualizace probíhá načtením nového zdrojového souboru do seznamu řádků `List<nLine>` a stažením příslušného XML dokumentu ze serveru. Z XML dokumentu se načtou řádky původního zdrojového souboru do seznamu `List<oLine>`. Oba seznamy řádků se pošlou do Komparátoru, kde se porovnají a shodné řádky se z obou seznamů odstraní. Komparátor pak vrací dva seznamy řádků. V seznamu `List<nLine>` jsou uloženy řádky, které původní zdrojový soubor neobsahuje, jsou tedy přidané. A seznam `List<oLine>` obsahuje řádky, které v původním zdrojovém souboru jsou, ale v novém nikoli, tedy řádky, které byli odstraněny. XML parser oba seznamy zpracuje a upraví podle nich XML dokument, který poté odešle na server.

Pro načtení uloženého projektu se stáhne příslušný XML dokument ze serveru a XML parser z něj vygeneruje zdrojový soubor nebo některou z jeho předchozích verzí. Aplikace bude umožňovat, vygenerovat zdrojový soubor bez určité změny, nebo vygenerovat pouze změny od určitého uživatele, vždy však tak, aby byl výsledný zdrojový soubor přeložitelný.

Aplikace bude také ukládat komentáře k jednotlivým změnám zdrojového souboru, pokud je bude uživatel komentovat. Komentování změn je nepovinné. Další z funkcí aplikace jsou, tisk seznamu projektů uložených na serveru a tisk seznamu změn či verzí v rámci jednoho projektu na standardní výstup.

4.1.2 Server

Jako úložiště projektů, dostupné pro všechny uživatele aplikace, bude použito Subversion serveru. Protože správu verzí řeší sama aplikace, nebude u služby Subversion využita žádná její vlastnost týkající se verzování, ale bude sloužit pouze pro uložení dat. To má výhodu v tom, že aplikace není na použití Subversion nijak závislá a bylo by možné využít jako úložiště dat jakoukoliv jinou podobnou službu, či vzdálený server.

Se Subversion serverem aplikace komunikuje pomocí metod a rozhraní open-source knihovny SVNkit. Aplikace si stáhne pracovní kopii repositorů, v té se provádí přislušné aktualizace či přidávání nových souborů a na konci operace se změny odešlou na Subversion server.

4.2 Metoda udržování záznamu o změnách

Pro ukládání zdrojových souborů LaTeXu a zaznamenávání změn a verzí, jsem zvolil jazyk XML. Hlavním důvodem zvolení XML byla možnost vytvoření vlastní struktury archivování verzí, snadné zpracovávání dokumentů pomocí knihoven Javy a přehlednost a čitelnost XML dokumentů i pro člověka.

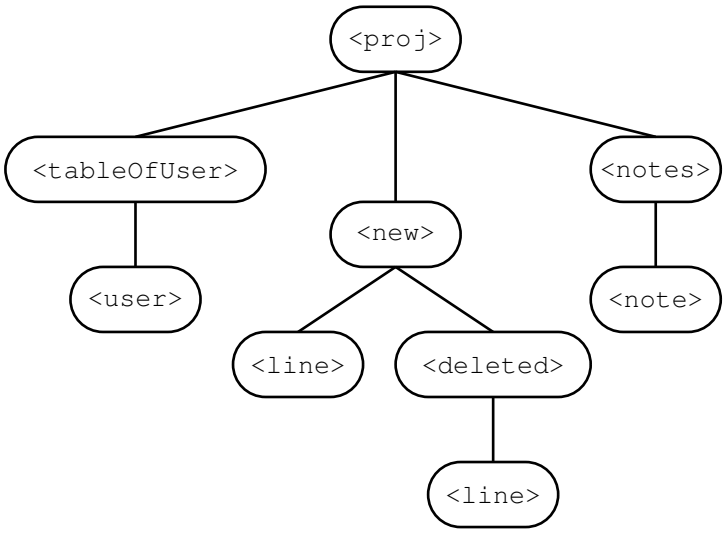
4.2.1 Požadavky

Při návrhu struktury XML, bylo nutné vzít v potaz tyto požadavky kladené na aplikaci. Uživatel se do aplikace nepřihlašuje, ale je identifikován podle jména uživatele počítače, z něhož aplikaci spouští. Aplikace podporuje správu více uživatelů, každý uživatel může přidávat nový text a měnit jakoukoliv část dokumentu. Aplikace neřeší správnost LaTeXového souboru. Aplikace je schopna vygenerovat z XML dokumentu, jen změny určitých uživatelů, tikne na výstup seznam jednotlivých změn a informace o nich. Samozřejmostí je návrat k předchozím verzím. Při zpracovávání nové verze LaTeXového souboru, se bude nový soubor porovnávat s obsahem XML, proto je nutné, aby šla z XML dokumentu jednoduše vygenerovat poslední verze zdrojového souboru.

4.2.2 Popis struktury XML dokumentu

Zdrojový soubor LaTeXu a všechny jeho změny a předchozí verze budou uloženy v jednom XML dokumentu, jméno tohoto dokumentu se generuje ze jména LaTeXového souboru, a až na příponu je tedy stejné. V první fázi je soubor identifikován podle svého jména, ve fázi druhé podle atributů kořenového elementu `<proj>`. Atribut `date` obsahuje datum vytvoření projektu a v atributu `name` je obsaženo celé jméno a přípona LaTeXového souboru. Dalším atributem kořenového elementu je atribut `status`, který označuje aktuální stav dokumentu a může nabývat tří hodnot `open` – dokument je otevřený a každý uživatel ho může editovat, `locked` – dokument je zamčený a nemůže být editován, `deleted` – dokument je smazán, avšak fyzicky stále existuje a je možno ho obnovit. Atribut status může měnit uživatel-správce, implicitně je nastaven status `open`.

Uživatelé jsou definováni v elementu `<tableOfUser>`, který obsahuje pro každého uživatele element `<user>` s atributy `name` a `userID`. Protože se uživatelé nepřihlašují, ale jsou identifikováni jménem uživatele počítače, na kterém je aplikace spuštěna, je toto jméno obsaženo v atributu `name`. Atribut `userID` obsahuje jednoznačné identifikační číslo uživatele, kterým budou také označeny všechny změny, které uživatel udělal. Uživatel, který vytvoří nový dokument je uživatel-správce a je označen nejnižším indetifikačním číslem.



Obrázek 4.2: Objektová hierarchie XML dokumentu

Jednotlivé změny jsou obsaženy v elementech `<new>` a `<deleted>`. Element `<deleted>` je potomkem elementu `<new>`, kdyby tomu tak nebylo, elementy `<new>` by byly přepisovány elementy `<deleted>` a nebylo by možno se pohybovat ve verzích. Při porovnávání s novou verzí zdrojového souboru, bude nový text, neobsažený v původní verzi označen elementem `<new>`. Naopak text, který je v původní verzi souboru, ale nová verze ho neobsahuje bude označen elementem `<deleted>`. Každá změna LaTeXového souboru má své identifikační číslo, elementy vytvořené v průběhu jednoho porovnávání mají stejné identifikační číslo. První verzi dokumentu bude zapouzdřovat element `<new>` s nejnižším identifikačním číslem. Každá další změna bude mít identifikační číslo o jednu vyšší. Aby bylo možno efektivně se pohybovat v předchozích verzích, je řada identifikačních čísel pro elementy `<new>` a `<deleted>` společná. Dále obsahují elementy `<new>` a `<deleted>` atribut `userID`, pomocí něhož jsou elementy identifikovány s uživatelem, který je vytvořil a atribut `dateTime`, který obsahuje datum a čas provedení změny.

Nové verze LaTeXového souboru se porovnávají po řádcích. Aby bylo možno efektivně vy-

volávat řádky z XML dokumentu přímo k porovnávání, je každý řádek zapouzdřen elementem <line>. Element <line> má atribut cl (count line), který obsahuje číslo řádku, pro lepší orientaci v XML dokumentu při přidávání nových elementů <new> a <deleted>. Po aktualizaci XML dokumentu dochází k přečíslování řádků.

V elementu <notes> a jeho potomcích <note> jsou uchovány komentáře k jednotlivým změnám, ke každé změně náleží právě jeden komentář a komentáře jsou propojeny se změnami identifikačním číslem, které je obsaženo v atributu <id>. Komentáře jsou nepovinné a budou se spolu s identifikačním číslem, datem provedení a autorem změny tisknout na výstup, když uživatel zvolí možnost zobrazit seznam změn.

Na následujícím jednoduchém zdrojovém souboru LaTeXu si ukážeme, jak bude vypadat jeho reprezentace v XML dokumentu a předvedeme si jak se XML dokument změní při aktualizaci LaTeXového souboru.

```
%název souboru = priklad.tex, uživatel = xdivil00
\documentclass[a4paper,12pt]{report}
\begin{document}

\section{Ukázový soubor}

Tento soubor je demonstrační.

\end{document}
```

Níže je uveden dokument XML, který vznikl z předchozího zdrojového LaTeXového souboru. Element <notes> je prázdný, to znamená, že zdrojový soubor byl uložen bez komentáře. Prázdné řádky ze zdrojového souboru se v XML dokumentu změní v prázdné elementy <line/>.

```
<proj name="priklad.tex" date="15.4.2010" status="open">
  <tableOfUser>
    <user name="xdivil00" userID="1"/>
  </tableOfUser>
  <notes/>
  <new id="1" dateTime="15.4.2010 21:16:28" userID="1">
    <line cl="1">%název souboru=priklad.tex, uživatel=xdivil00</line>
    <line cl="2">\documentclass[a4paper,12pt]{report}</line>
    <line cl="3">\begin{document}</line>
    <line cl="4"/>
    <line cl="5">\section{Ukázový soubor}</line>
    <line cl="6"/>
    <line cl="7">Tento soubor je demonstrační.</line>
    <line cl="8"/>
    <line cl="9">\end{document}</line>
  </new>
</proj>
```

Dále si převedeme jak se změní XML dokument, pokud změníme zdrojový soubor následujícím způsobem a změnu řádně okomentujeme.

```
...
Tento soubor je demonstrační.
Připíšeme nový řádek na konec souboru.
A ještě jeden, aby toho nebylo málo.

\end{document}
```

Stávající element <new id="1"> se rozdělil v místě přidání nových řádků a na toto místo byl vložen nový element <new id="2">, tyto řádky obsahující. Také element <notes> obsahuje potomka <note> s komentářem, který má identifikační číslo totožné s identifikačním číslem nově přidaného elementu <new>. Elementy <line> za novým elementem <new id="2"> byly přečíslovány.

```
<proj name="priklad.tex" date="15.4.2010" status="open">
 <tableOfUser>
  <user name="xdivil00" userID="1"/>
 </tableOfUser>
 <notes>
  <note id="2">Pridani dvou radku</note>
 </notes>
 <new id="1" dateTime="15.4.2010 21:16:28" userID="1">

  ...

  <line cl="7">Tento soubor je demonstrační.</line>
 </new>
 <new id="2" dateTime="15.4.2010 21:53:59" userID="1">
  <line cl="8">Připíšeme nový řádek na konec souboru.</line>
  <line cl="9">A ještě jeden, aby toho nebylo málo.</line>
 </new>
 <new id="1" dateTime="15.4.2010 21:16:28" userID="1">
  <line cl="10"/>
  <line cl="11">\end{document}</line>
 </new>
</proj>
```

V posledním příkladu předvedeme jak se projeví vymazání sedmého a osmého řádku a přepsání řádku pátého. Změnu bude provádět jiný uživatel uzivatel. Aktualizovaný zdrojový soubor vypadá následovně:

```
%název souboru = priklad.tex, uživatel = xdivil00
\documentclass[a4paper,12pt]{report}
\begin{document}

\section{Torzo ukázového souboru}

A ještě jeden, aby toho nebylo málo.

\end{document}
```

Nový uživatel byl uložen v elementu <tableOfUser> a má identifikační číslo id=2. Řádky, které aktualizovaný soubor neobsahuje, jsou uloženy v elementech <deleted>. Nesmazané řádky a elementy <deleted> se zapouzdřují do původních elementů <new>. I když pátý řádek nebyl vymazán, ale jen přepsán, byla jeho původní verze uložena v elementu <deleted> a nová verze byla vložena na jeho místo v novém elementu <new>. Všechny elementy <line>, jejichž přímým rodičem je element <new>, byly přečíslovány.

```
<proj name="priklad.tex" date="15.4.2010" status="open">
 <tableOfUser>
```

```
  <user name="xdivil00" userID="1"/>
  <user name="uzivatel" userID="2"/>
 </tableOfUser>
 <notes>
  <note id="2">Pridani dvou radku</note>
  <note id="3">Finalni uprava</note>
 </notes>
 <new id="1" dateTime="15.4.2010 21:16:28" userID="1">
  <line cl="1">% název souboru=priklad.tex, uživatel=xdivil00</line>
  <line cl="2">\documentclass[a4paper,12pt]{report}</line>
  <line cl="3">\begin{document}</line>
  <line cl="4"/>
  <deleted id="3" dateTime="15.4.2010 22:21:06" userID="2">
   <line cl="5">\section{Ukázový soubor}</line>
  </deleted>
 </new>
 <new id="3" dateTime="15.4.2010 22:21:06" userID="2">
  <line cl="5">\section{Torzo ukázového souboru}</line>
 </new>
 <new id="1" dateTime="15.4.2010 21:16:28" userID="1">
  <line cl="6">
  <deleted id="3" dateTime="15.4.2010 22:21:06" userID="2">
   <line cl="7">Tento soubor je demonstrační.</line>
  </deleted>
 </new>
 <new id="2" dateTime="15.4.2010 21:53:59" userID="1">
  <deleted id="3" dateTime="15.4.2010 22:21:06" userID="2">
   <line cl="8">Připíšeme nový řádek na konec souboru.</line>
  </deleted>
  <line cl="7">A ještě jeden, aby toho nebylo málo.</line>
 </new>
 <new id="1" dateTime="15.4.2010 21:16:28" userID="1">
  <line cl="8"/>
  <line cl="9">\end{document}</line>
 </new>
</proj>
```

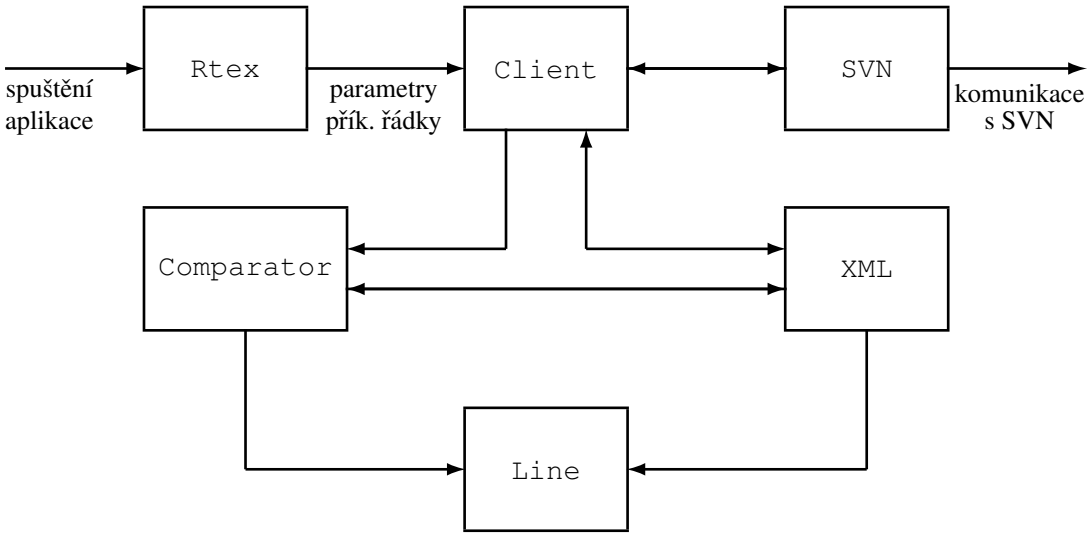
Takto navržená metoda je efektivní, zejména z hlediska generování zdrojových souborů LaTeXu a je paměťově nenáročná, což je výhodou i proto, že při zpracovávání XML dokumentů bude používám objektový model DOM.

Kapitola 5

Popis řešení

Tato kapitola popisuje implementaci výsledné aplikace a její funkcionalitu.

5.1 Implementace klientské aplikace



Obrázek 5.1: Diagram tříd

Při spuštění aplikace se vytvoří instance třídy `Rtex`, která načte parametry příkazového řádku a předá je třídě `Client`, jejíž instanci vytvoří. V instanci třídy `Client` se zpracovávají parametry příkazové řádky, které určují další běh aplikace. Instance třídy `Client` také při každém spuštění načte data z konfiguračního souboru `conf.xml`. Konfigurační soubor je XML dokument, který obsahuje cestu k adresáři se zdrojového soubory LaTeXu, adresu používaného SVN, uživatelské jméno a heslo pro přístup k SVN.

Struktura konfiguračního souboru vypadá následovně:

```
<conf>
<scpath>C:/Documents and Settings/Jarosh/Plocha/tex_zdr/</scpath>
<svnurl>svn://merlin.fit.vutbr.cz:3691</svnurl>
<username>rtex</username>
<password>rt3x</password>
</conf>
```

Třída `XML` obsahuje metody, které pracují s XML dokumenty a používá k tomu objektový model DOM. Vytváří nové XML dokumenty, zaznamenává změny v již vytvořených XML dokumentech a generuje z XML dokumentů zpětně zdrojové soubory LaTeXu.

Třída `Comparator` porovnává dva seznamy instancí třídy `Line`, které zapouzdřují obsah řádku (řetězec `String`) a jeho pořadové číslo (celé číslo `Integer`) ve zdrojovém souboru.

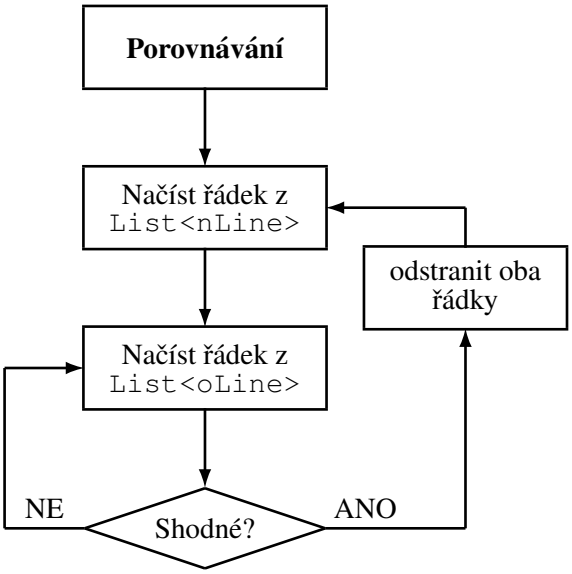
Třída `SVN` obstarává komunikaci se Subversion serverem pomocí knihovny SVNkit. Obsahuje metody, pro stáhnutí repozitářů ze SVN serveru, uložení aktualizovaných souborů a vytvoření nových souborů na SVN serveru.

V následujících kapitolách popíšeme, jak pracují jednotlivé funkce aplikace.

5.1.1 Nový projekt

Při úplně prvním spuštění aplikace se vytvoří v adresáři s aplikací adresář `repository` a do něj se stáhne pracovní kopie používaného Subversion repozitáře. Aktuální obsah se ze SVN serveru stahuje, při každém spuštění aplikace. Aplikace zjistí zda projekt stejného jména už není vytvořen v pracovní kopii, pokud není načte se metodou třídy XML zdrojový soubor do dočasného XML dokumentu, který se metodou třídy SVN odešle na SVN server. V pracovní kopii se změna projeví až s dalším spuštění aplikace, kdy se opětovně aktualizuje obsah pracovní kopie.

5.1.2 Aktualizace projektu



Obrázek 5.2: Vývojový diagram porovnávání řádků

Aktualizovaný zdrojový soubor se po řádcích načte do seznamu instancí třídy `Line`, `List<nLine>`. Původní zdrojový soubor je uložen v adresáři `repository` a metodou třídy `XML` se z něj vygenerují do seznamu instancí třídy `Line`, `List<oLine>` obsahy elementů `<line>` poslední verze zdrojového souboru. Oba seznamy se pak předají instanci třídy `Comparator`, kde proběhne jejich porovnání. Algoritmus porovnávání je zobrazen na obrázku 5.2, porovnávají se pouze obsahy řádků, tedy řetězce. Po skončení porovnávání zůstanou v obou seznamech jen řádky, pro které nebyla nalezena shoda. Oba seznamy se předají instanci třídy `XML`.

V instanci třídy `XML` se provede aktualizace obsahu dokumentu. Seznam `List<oLine>` obsahuje řádky, které aktulizovaný soubor neobsahuje. Tyto řádky se najdou v XML dokumentu a metoda `updateXML()` je označí jako smazané. Řádky v seznamu `List<nLine>` jsou nové řádky, pro tyto řádky se najde příslušné místo v XML dokumentu, tak aby pořadí řádků v XML dokumentu odpovídalo pořadí řádků v LaTeXovém souboru. Aktualizovaný XML dokument se uloží a odešle do SVN repozitáře.

Metodu `updateXML()` popíšeme podrobněji. Metoda je složena ze dvou částí, jedna část řeší přidávání textu a druhá mazání textu. Při přidávání textu, je vytvořen element `<new>` a do něj jsou ze seznamu `List<nLine>` přidány všechny řádky, které na sebe bezprostředně navazují. Po vložení řádku do elementu `<new>` se řádek vymaže ze seznamu `List<nLine>`. Následně se pomocí čísel řádků vyhledá místo, kam nový blok řádků patří, v tomto místě se stávající element `<new>` rozdělí a na vzniklé místo se vloží nový element. Po přidání nového elementu se přepíší čísla řádků, aby odpovídala svému pořadí. Pokud jsou v seznamu `List<nLine>` další řádky pokračuje se stejným způsobem dál. Přepisování čísel řádků, po každém vložení elementu `<new>` má velký význam, pokud by k přepisu čísel řádků nedocházelo, nebylo by nalezeno správné místo pro vložení dalšího elementu `<new>` a nemohlo by být zachováno správné pořadí řádků. To by byl, z hlediska funkčnosti aplikace, velký problém.

Postup při mazání řádků je vzhledem k tomu, že elementy `<deleted>` jsou potomci elementů `<new>`, poněkud jednodušší. Element se vytvoří, podle pořadí řádků se vloží na místo kam patří, smazané řádky jsou do něj přesunuty a pokračuje se dál. Po této operaci se, ale narozdíl od přidávání řádků, řádky přečíslovávat nesmějí. A to ze stejného důvodu, pokud by byli čísla řádků přepsány (Přepisují se jen čísla řádků, které jsou obsaženy v aktuální verzi, to znamená, že smazané řádky se neberou v potaz.) nebylo by nalezeno správné místo pro umístění dalšího elementu a do elementu by se vložili jiné řádky. Čísla řádků se tedy upravují až na závěr, kdy jsou všechny elementy `<deleted>` na svém místě.

Změna číslování řádků má ještě jedno úskalí. Pokud jsou v jedné aktualizaci řádky zároveň přidány i smazány je nutné neprve zpracovat řádky smazané, přepsat čísla řádků a následně zpracovat přidané řádky. Stejná situace nastává i pokud je nějaký řádek přepsán, aplikace totiž přepis řádku chápe tak, že jeden řádek byl smazán a místo něj byl přidán řádek jiný.

5.1.3 Načtení projektu

Načtení projektu má několik variant. V první řadě je možno vyvolat poslední verzi projektu. Další možností je vygenerovat některou předchozí verzi projektu a tuto volbu je možné kombinovat s volbou vygenerování LaTeXového souboru bez některé dřívější změny. Zdrojové soubory LaTeXu se generují z XML souborů uložených v repozitářích. V obou případech se vygenerovaným LaTeXovým souborem přepisuje původní soubor v adresáři určeném cestou zadanou v konfiguračním souboru.

To co bylo problémem při ukládání změn a aktualizování projektů, tedy zachování správného pořadí řádků v XML dokumentu, je nyní výhodou. Protože jsou řádky v dokumentech XML poskládány ve správném pořadí, stačí při generování zdrojových souborů načítat řádky tak, jak jsou uloženy, od prvního k poslednímu a pouze kontrolovat zda splňují podmínky, aby mohly být načteny.

Dříve než podrobněji popíši jak fungují varianty načítání projektu, bylo by vhodné ujasnit si termíny změna a verze projektu v souvislosti s dokumentem XML. Změna jsou všechny elementy `<new>` a `<deleted>` se stejným identifikačním číslem. Verzí projektu se rozumí všechny změny s identifikačním číslem nižším a stejným jako je číslo verze. Princip generování identifikačních čísel je popsán v kapitole 4.2.2.

Vygenerování poslední verze, což bude při spolupráci více autorů asi nejčastější volba je nej-

jednoduší. Z XML dokumentu se načtou pouze všechny řádky, jejichž přímým rodičem je element `<new>`.

Pokud chceme vygenerovat některou předchozí verzi, načteme elementy `<new>`, které mají identifikační číslo stejné a nižší než požadovaná verze a elementy `<deleted>` v nich obsažené, které mají identifikační číslo vyšší než číslo požadované verze. Tedy, chceme-li dostat pátou verzi projektu, načteme řádky z elementů `<new>`, které mají `id = 1`, `2`, `3`, `4` a `5` a pokud jsou v těchto elementech řádky, které jsou přímými potomky elementu `<deleted>`, načítáme je pouze v případě, že jejich `id > 5`, to znamená řádky, které byli vymazány až v pozdějších verzích.

Při vygenerování projektu bez určité změny, například bez třetí změny, se nebudou načítat řádky z elementů `<new>`, které mají `id = 3` a naopak řádky jejichž rodičem jsou elementy `<deleted>` s `id = 3` se načítat budou.

Poslední možností je vygenerovat z XML dokumentu jen změny vytvořené jedním uživatel. Tyto změny se vygenerují do LaTeXového souboru, tak aby byl přeložitelný. Pokud má být výsledný soubor přeložitelný je potřeba aby byla načtena i preambule, proto se nejprve načítají všechny řádky bez rozdílu, až po řádek značící začátek dokumentu včetně ho, potom se načítají pouze řádky jejichž autorem je zvolený uživatel a na konci souboru, pokud není poslední řádek, řádkem značící konec dokumentu, se tento vloží. Změny ve kterých uživatel mazal text z původního LaTeXového souboru budou ve vygenerovaném LaTeXovém souboru pro odlišení zakomentovány.

5.1.4 Další funkce

Další funkce nepracují přímo z obsahem XML dokumentů. Jsou to fukce pro vypsání seznamu projektů v `repository` adresáři na standardní výstup ve formátu, jméno souboru, datum založení, stav souboru a zakladatel. Například:

```
proj.tex  3.2.2010  open  xdivil00
```

A funkce, která vypíše na standardní výstup seznam změn v jednom projektu, data změn a jejich autory v tomto formátu.

```
Name: proj.tex
Created: 3.2.2010
Status: open

ID: Date of change:     Author:  Description:
1   3.2.2010 16:32:20   xdivil00 Vytvoreni souboru
2   4.2.1010 09:56:32   xdivil00 Pridani dalsiho odstavce
```

Mezi další funkce patří i možnosti správce, tedy zakladatele projektu. Správce má ještě k dispozici, funkce pro zamčení projektu, odemčení projektu a vymazání projektu. Pokud je projekt zamčen je nepřístupný uživatelům pro jakékoliv úpravy či aktualizace, ale může být stahován a můžou být generovány jeho předchozí verze a podobně, jinými slovy zamčený projekt je přístupný pouze ke čtení. Zamčení projektu je vratná změna a správce může projekt znovu odemčít. Pokud je projekt vymazán není uživatelům přístupný ani pro čtení ani pro zápis. Vymazání projektu je změna nevratná. Z projektů označených jako `deleted` může číst pouze jejich správce a má tak možnost projekt pro ostatní uživatele obnovit, ovšem bez zachování jeho historie.

Kapitola 6

Závěr

Zadání práce bylo splněno. Sázecí systém LaTeX jsem používal již dříve a syntaxi jeho zdrojových souborů jsem dobře znal. Seznámil jsem se s různými typy revizních systémů a jejich problematikou. Na základě toho jsem navrhnul jednoduchou a paměťově nenáročnou metodu udržování revizí zdrojových souborů systému LaTeX v dokumentech XML. Tuto metodu jsem implementoval ve výsledné aplikaci. Výsledná aplikace má všechny požadované vlastnosti a podporuje správu více uživatelů.

Aplikaci by bylo možné rozšířit o jednoduché grafické uživatelské rozhraní, které by zjednodušilo ovládání aplikace a více by ji tak zpřístupnilo běžnému uživateli. Aplikace v současnosti dokáže zpracovat jeden zdrojový soubor LaTeXu a předpokládá, že je tento soubor přeložitelný. Do budoucna by bylo možné se zamyslet nad rozšířením, které by umožňovalo správu rozsáhlejších projektů.

Za přínos této práce považuji seznámení se s problematikou revizních systémů. Získal jsem znalosti o dnešních systémech pro správu verzí, o jejich možnostech a vlastnostech a seznámil jsem se s pokročilými algoritmy, které používají.

Zhodnocení jednotlivých bodů zadání

O prvním bodu zadání, seznámení se s sázecím sytémem LaTeX a jeho syntaxí, pojednává kapitola 3.1. Navrženou metodu udržování záznamů o změnách prováděných v dokumentu, umožnující návrat k předchozím verzím, odstraňování jednotlivých změn a komentování částí textu jsem podrobně popsal v kapitole 4.2. Návrh systému jsem popsal v kapitole 4.1 a popisem jeho implementace a fukčnosti se zabývá kapitola 5. O dosažených výsledcích a možných rozšířeních aplikace pojednává tato kapitola.

Literatura

[1] Baudiš, P.: Výlet do říše verzí. 2004, [online], [cit. 2010-05-07].
URL http://www.root.cz/clanky/vylet-do-rise-verzi/

[2] Outrata, J.: Version Control Systems. 2007, [online],[cit. 2010-05-07].
URL phoenix.inf.upol.cz/~outrata/download/texts/VCS-slides.pdf

[3] Spell, B.: Java Programujeme profesionálně. Computer Press, 2002, iSBN 80-7226-667-5.

[4] WWW stránky : SVNkit. [online], [cit.2010-05-07].
URL http://svnkit.com/index.html

[5] Oetiker T.; Partl H.; Hyna I.; aj.: Ne příliš stručný úvod do systému LaTeX 2_ε. 1998, [online], [cit.2010-04-23].
URL http://www.penguin.cz/ kocer/texty/lshort2e/lshort2e-cz.pdf

[6] Křena B.: Typografie a publikování ITY 2008/2009. VUT Brno.

[7] Wikipedia: Apache Subversion – Wikipedie, free encyclopedia. [online], [cit. 2010-05-07].
URL http://en.wikipedia.org/wiki/Apache_Subversion

[8] Wikipedia: Concurrent Versions System – Wikipedia, free encyclopedia. [online], [cit.2010-05-07].
URL http://en.wikipedia.org/wiki/Concurrent_Versions_System

[9] Wikipedia: Revision Control System – Wikipedia, free encyclopedia. [online], [cit.2010-05-07]
URL http://en.wikipedia.org/wiki/Revision_Control_System

[10] Wikipedia: Source Code Control System – Wikipedia, free encyclopedia. [online], [cit.2010-05-07]
URL http://en.wikipedia.org/wiki/Source_Code_Control_System

[11] Wikipedie: GNU Arch – Wikipedie, free encyclopedia. [online], [cit. 2010-05-07].
URL http://en.wikipedia.org/wiki/GNU_arch

[12] Wikipedie: SVK – Wikipedie, free encyclopedia. [online], [cit. 2010-05-07].
URL http://en.wikipedia.org/wiki/SVK

Seznam obrázků

Dodatek A

Uživatelská příručka

A.1 Instalace

Pro spuštění aplikace stačí překopírovat adresář `rtex/dist` z přiloženého CD na cílový počítač a v tomto adresáři aplikaci spustit příkazem `java -jar "rtex.jar" -parametry aplikace`. Na počítači, na kterém je aplikace spouštěna, musí být nainstalován virtuální stroj jazyka Java (JVM).

Před prvním spuštěním je nutné nastavit adresu používaného Subversion serveru, uživatelské jméno, heslo a cestu k adresáři se zdrojovými soubory, které chceme spravovat, v souboru `conf.xml`, který se nachází v adresáři `dist`.

A.2 Ovládání aplikace

Aplikace se ovládá pomocí parametrů příkazového řádku, které upřesňující zvolenou funkci aplikace. Pokud se aplikace spustí bez parametrů, tak se pouze aktualizuje pracovní kopie repozitáře.

Uložení nového projektu se provádí spuštěním aplikace s parametrem `-commit`, následovaným jménem ukládaného souboru a případným komentářem uvedeným do uvozovek.

```
-commit proj.tex "novy projekt"
```

Pro uložení aktualizace projektu je nutno spustit aplikaci s parametrem `-update`, jinak má příkaz stejný formát jako příkaz pro uložení nového projektu.

```
-update proj.tex "aktualizovany text"
```

Příkaz pro načítání projektů má více variant. Zakladní varianta, která vypadá následovně,

```
-load proj.tex
```

vygeneruje poslední verzi projektu. Abychom vygenerovali některou předchozí verzi, rozšíříme příkaz o parametr `-v` a číslo verze, například `-v 2`, což znamená, že chceme vygeneroval druhou verzi projektu. Dále je možno vygenerovat projekt bez určité změny, to se provádí parametrem `-w` (z anglického without – bez) následovaným číslem změny, například příkaz s parametrem `-w 3`, vygeneruje aktuální verzi bez změny číslo tři. Parametry pro vygenerování předchozí verze a pro vygenerování projektu bez určité změny se mohou použít zároveň, ale parametr `-v` musí být vždy uveden jako první. Níže jsou uvedeny příklady kompletních příkazů.

```
-load proj.tex -v 2
-load proj.tex -w 3
-load proj.tex -v 5 -w 2
```

Poslední modifikace příkazu je užití parametru `-u jmeno` (`-u` jako uživatel) pro vygenerování jen změn, jejichž autorem je určitý uživatel. Celý parametr pak vypadá následovně.

```
-load proj.tex -u xdivil00
```

Vytisknutí seznamu souborů uložených v repozitářích je možno vytisknout spuštěním aplikace s parametrem:

```
-listOfFiles
```

Výpis jednotlivých změn projektu nebo-li jeho historii dostaneme při spuštění aplikace s parametrem `-versions` a názvem souboru, jehož historii chceme vytisknout.

```
-versions proj.tex
```

Správce projektu může ještě použít parametry pro zamčení projektu, odemčení projektu a smazání projektu.

```
-lock proj.tex
-unlock proj.tex
-delete proj.tex
```

Dodatek B

Obsah CD

- **`rtex`** – adresář obsahuje samotnou aplikaci.
  - **`src`** – adresář obsahuje zdrojové soubory aplikace.
    - `Rtex.java`
    - `Client.java`
    - `XML.java`
    - `SVN.java`
    - `Comparator.java`
    - `Line.java`
  - **`dist`** – adresář obsahuje spustitelný JAR archiv `rtex.jar`, potřebné knihovny v adresáři `lib` a konfigurační soubor `conf.xml`.
- **`zprava`** – tento adresář obsahuje zdrojové soubory technické zprávy a další soubory potřebné pro přeložení. Také obsahuje již přeloženou zprávu v souboru `projekt.pdf`.
- **`readme.txt`** – soubor obsahuje základní informace k instalaci a spuštění aplikace a popis obsahu CD.